



兰州大学

毕业论文

(本科生)

论文题目（中文） 《数据结构》课程的学习报告

论文题目（英文） Study Report of Data Structures

学 生 姓 名 谭源

导师姓名、职称 蒙应杰

学生所属学院 信息科学与工程学院

专 业 计算机类

年 级 2019 级

兰州大学教务处

诚信责任书

本人郑重声明：本人所呈交的毕业论文（设计），是在导师的指导下独立进行研究所取得的成果。毕业论文（设计）中凡引用他人已经发表或未发表的成果、数据、观点等，均已明确注明出处。除文中已经注明引用的内容外，不包含任何其他个人或集体已经发表或在网上发表的论文。

本声明的法律责任由本人承担。

论文作者签名：_____ 日期：_____

关于毕业论文（设计）使用授权的声明

本人在导师指导下所完成的论文及相关的职务作品，知识产权归属兰州大学。本人完全了解兰州大学有关保存、使用毕业论文（设计）的规定，同意学校保存或向国家有关部门或机构送交论文的纸质版和电子版，允许论文被查阅和借阅；本人授权兰州大学可以将本毕业论文（设计）的全部或部分内容编入有关数据库进行检索，可以采用任何复制手段保存和汇编本毕业论文（设计）。本人离校后发表、使用毕业论文（设计）或与该毕业论文（设计）直接相关的学术论文或成果时，第一署名单位仍然为兰州大学。

本毕业论文（设计）研究内容：

- 可以公开
- 不宜公开，已在学位办公室办理保密申请，解密后适用本授权书。

（请在以上选项内选择其中一项打“√”）

论文作者签名：_____ 导师签名：_____

日期：_____ 日期：_____

《数据结构》课程的学习报告

中文摘要

在计算机科学中，数据结构是计算机中存储、组织数据的方式。这学期蒙老师先由数据结构的定义开始讲起，依次讲解了算法、线性表、栈和队列、串、数组和广义表、树形结构、图结构、排序、数据检索这几部分内容

不要仅仅把它当做广告，这里面有很多 latex 的用法说明

关键词： 兰朵儿， i 兰大易班， yuh

STUDY REPORT OF DATA STRUCTRES

Abstract

This essay explores the history of studies in analytical philosophy in China since the beginning of the last century, by dividing into three phases. It shows that, in these phases, analytic philosophy was always at a disadvantage in confronting serious challenges coming from both Chinese traditional philosophy and modern philosophical trends. The authors argue that Chinese philosophers have both done preliminary studies and offered their own analyses of various problems as well as some new applications of analytic philosophy especially in the latest period. Meanwhile, Chinese traditional philosophy was always trying to adjust its cultural mentality in the struggle with analytic philosophy, and accommodated in its own way the rationalistic spirit and scientific method represented in analytic philosophy.

Key Words: analytical philosophy; Chinese philosophers; philosophical analysis; dialogue in philosophy.

目 录

中文摘要	I
英文摘要	II
第一章 数据结构绪论	1
1.1 数据结构的基本概念及研究内容	1
1.2 数据结构的选择与评价	2
第二章 算法	3
2.1 算法的定义	3
2.1.1 概述	3
2.1.2 定义	3
2.1.3 内涵及分类	3
2.1.4 算法与程序的异同	3
2.2 算法的描述及设计原则	3
2.2.1 算法描述方法	3
2.2.2 算法基本标准	4
2.3 算法分析概论及有效算法	4
2.3.1 概念	4
2.3.2 复杂性分析	6
2.4 算法设计方法概论	7
2.4.1 概述	7
2.4.2 算法设计的基本技术	7
2.5 算法描述语言	7
2.5.1 PDL 概述	7
2.5.2 PDL 的优势	8
2.5.3 PDL 书写及要求	8

第三章 线性表	13
3.1 线性表及其运算	13
3.1.1 线性表的定义	13
3.1.2 线性表的特征	13
3.1.3 线性表的运算	13
3.2 线性表的储存表示	13
第四章 栈和队列	17
4.1 栈及其运算	17
4.1.1 绪论	17
4.1.2 栈的基本定义	17
4.1.3 与线性表的关系	17
4.1.4 栈的运算	17
4.1.5 栈的存储与运算的实现	17
4.1.6 多栈共存问题	17
4.2 栈的应用	17
4.3 队列及其运算	17
4.4 受限的栈及队列（了解）	18
第五章 串	19
5.1 串及其运算	19
5.2 串的模式匹配	20
5.3 例题	21
第六章 数组和广义表	25
第七章 树形结构	26
7.1 树的基本定义和运算	26
7.1.1 树形结构的概述	26
7.1.2 树的基本定义	26
7.1.3 有序树的基本定义	26
7.1.4 森林（树林）	27

7.2	二叉树	27
7.3	遍历二叉树	27
7.3.1	概念	27
7.3.2	二叉树的遍历次序 (order)	27
7.4	树、森林与二叉树的转换	28
7.4.1	概述	28
7.4.2	树转为二叉树	28
7.4.3	二叉树还原为树	29
7.4.4	森林转为二叉树	29
7.4.5	二叉树转为森林	30
7.4.6	树的遍历	30
7.4.7	森林的遍历	31
7.5	线索树	32
7.6	树形结构的应用	32
	第八章 图结构	33
8.1	基本概念	33
8.1.1	概述	33
8.1.2	概念	33
8.2	图的存储	37
8.2.1	邻接矩阵	37
8.2.2	邻接表	37
8.2.3	邻接多重表	39
8.3	图的遍历	40
8.3.1	概述	40
8.3.2	图的深度优先遍历	40
8.3.3	图的广度优先遍历	41
8.4	连通性及最小生成树	41
8.4.1	连通分量和生成树	41
8.4.2	最小生成树	41
8.5	有向无环图及应用	42
8.6	最短路径	42

第九章 latex 部分用法简介.....	43
9.1 用 latex 需要安装什么	43
9.1.1 texlive 下载安装	43
9.1.2 安装 IDE.....	44
9.2 常用的一些东西	45
9.2.1 国际三线表格.....	45
9.2.2 换页表格	46
9.2.3 字体	47
9.2.4 公式	47
9.2.5 左边大括号	48
9.2.6 复杂公式	48
9.2.7 等号对齐站	48
9.2.8 矩阵乘法	48
9.2.9 图, 并列排.....	48
9.2.10 附页代码	49
9.2.11 参考文献	50
参考文献	51
附录	51
致谢	52
论文(设计)成绩	53

第一章 数据结构绪论

其实我推荐绪论写在正文里!! 作为第一章

这里是绪论，也可以说是引言，在 LZUThesis2020.clc 里面改，引言写什么呢，先凑字数，

是真的在打广告啊，嗯，做兰大毕业论文 LaTex 模板时，顺便介绍一下我写的软件：i 兰大易班，兰大专属的 app，可以看课表，充值校园卡（微信、支付宝都可以），查成绩（可以算绩点），还可以……，好多好多好多

注意啊，段落在 latex 里面是要空一行的，不要简单一个回车

1.1 数据结构的基本概念及研究内容

数据元素：具有完整确定意义的描述现实的某一个客观实体的一个最小数据集。数据元素类似原子，可以再分，每一项被称作数据项

数据对象：具有相同属性的数据元素的集合

数据结构：给定数据对象及其上面定义的操作所共同构成的一个系统一个信息处理模型

主要研究的三个方面：

1. 数据的逻辑结构

- 逻辑关系

在自然形态下，数据元素之间的一种关系

- 逻辑结构

数据之间所有关系的一个集合数学表示： $B=(K,R)$ ，其中：K：数据上的有穷集合 R：K 上关系的有穷集合，其中每个关系 r 都是从 K 到 K 的关系

- 分类

– 线性

 一对一，单对单

– 非线性

 * 树形结构

 唯一一个直接前驱，多个直接后继

 * 图结构

2. 数据的存储结构

- 存储关系

存储关系的数学内涵：须要建立数据对象 (K) 到存储区域 (M) 的映射关系 (S):

S:KM

即 $k \in K$, 都有唯一的 $Z \in M$, 使得 $S(k)=Z$, Z 为 k 结点所占存储空间的始单元。

- 存储结构

- 顺序结构

按照连续地址空间的顺序依次的存放数据

- 链接结构

存储密度相比顺序结构下降

- 索引结构

- 散列结构

根据节点的值，通过一定的函数关系来确定数据元素的存储地址

3. 数据的运算关系定义在逻辑结构上，在存储结构上实施，即：

- 抽象层面

- 逻辑关系

- 需求

- 实现层面

- 存储关系

- 运算关系

- 评价层面

三个层次五个要素

1.2 数据结构的选择与评价

评价标准：

- 时间需要量时间效率
- 存储需要量存储效率

第二章 算法

2.1 算法的定义

2.1.1 概述

算法是一个问题的具体解决方案

2.1.2 定义

算法解决某一个问题的指令的有限集合

基本特征：

- 有穷性
- 确定性
- 可行性
- 输入
- 输出

2.1.3 内涵及分类

内涵体现在过程上

- 一般过程
- 函数过程

强调结果

2.1.4 算法与程序的异同

- 程序不是算法
- 程序不是算法程序不一定满足有穷性

2.2 算法的描述及设计原则

2.2.1 算法描述方法

- 计算机程序设计语言
优点与缺陷：设计出 = 实现出
- 自然语言缺陷：雍长二义性

- PDL
- 流程图

2.2.2 算法基本标准

- 正确性
- 易读性
- 健壮性
- 高效性

2.3 算法分析概论及有效算法

2.3.1 概念

一般不需要知道精确的时间消耗，需要知道时间消耗的增长率大体在什么范围。

算法复杂性的阶：算法比较主要比较阶

时间复杂性（时间渐进复杂性）：利用某算法处理一个问题规模为 n 的输入所需要的时间，记为 $T(n)$

空间复杂性：利用某算法处理一个问题规模为 n 的输入所需要的存储空间，记为 $S(n)$ 。

阶：对一个正常数 C ，一个算法在时间 (n^2) 内能处理规模为 n 的输入，则称此算法的时间复杂度是 (n^2) ，读作“ n^2 阶”，即该算法的时间复杂度与 n^2 是同阶的。

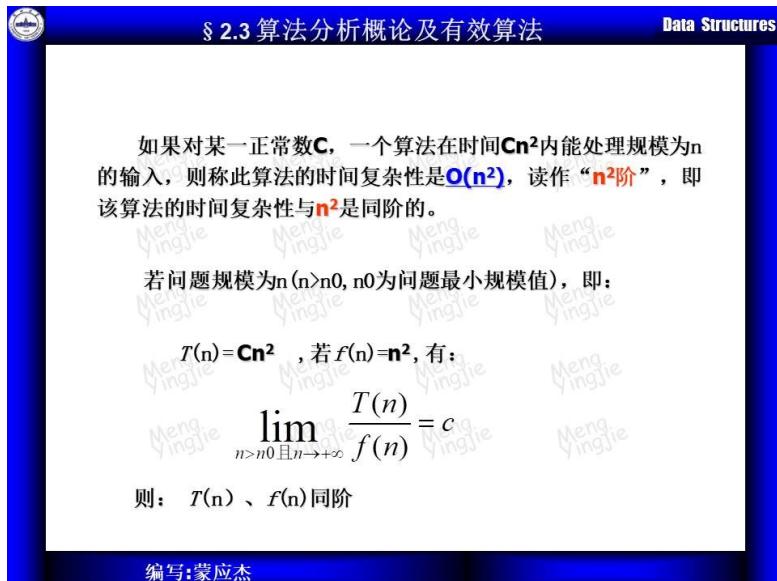


图 2.1 启动图形化安装界面

若一个算法时间复杂度为 $O(2n)$, 称其需要指数时间; 若是 $O(nk)$, 称其为多项式时间。当 n 非常大时两个时间差异非常大。

以多项式时间为界限的算法称为有效算法。如果一个问题不存在以多项式时间为界限的算法, 称为难解的 (难解性问题)

时间 复杂度	多项式时间算法与指数时间算法的运行时间对比					
	n=10	n=20	n=30	n=40	n=50	n=60
A算法	10^{-7}	$2*10^{-7}$	$3*10^{-7}$	$4*10^{-7}$	$5*10^{-7}$	$6*10^{-7}$
	n^2	10^{-6}	$4*10^{-6}$	$9*10^{-6}$	$1.6*10^{-5}$	$2.5*10^{-5}$
	n^3	10^{-5}	$8*10^{-5}$	$2.7*10^{-4}$	$6.4*10^{-4}$	$1.25*10^3$
	n^5	0.001	0.032	0.243	1.02	3.12
B算法	2^n	10^{-5}	0.001	10.74	3.05小时	130.3天
	3^n	$5.9*10^{-4}$	34.8	23.7天	3855年	$2*10^6$ 世纪
n 为数据规模, 在每秒运行速度为1亿次的计算机上的运行时间 (单位:秒)						

编写:蒙应杰

图 2.2 启动图形化安装界面

当 n 非常大时两个时间差异非常大。

The screenshot shows a software window titled "§ 2.3 算法分析概论及有效算法" (Algorithm Analysis Overview and Effective Algorithms) and "Data Structures". The main content is a table titled "提高计算机的速度对单位时间内可解的问题规模的影响" (The impact of increasing computer speed on the size of problems solvable within a unit time). The table compares two algorithms, A and B, across four columns representing different computer speeds: "现在的计算机" (Current Computer), "快100倍的计算机" (Computer 100 times faster), and "快1000倍的计算机" (Computer 1000 times faster). The table includes formulas for each row:

时间复杂度	单位时间内可解的问题规模		
	现在的计算机	快100倍的计算机	快1000倍的计算机
n	N_1	$100N_1$	$1000N_1$
n^2	N_2	$10N_2$	$31.6N_2$
n^3	N_3	$4.64N_3$	$10N_3$
n^5	N_4	$2.5N_4$	$3.98N_4$
2^n	N_5	$N_5 + 6.64$	$N_5 + 9.97$
3^n	N_6	$N_6 + 4.19$	$N_6 + 6.29$

Below the table, a note states: "从此表可看出, 把计算机速度提高1000倍, **A算法**在单位时间内可解问题的规模是原来的10倍, 而对于**B算法**, 问题规模只增加9.97 (≈ 10)个." (From this table, we can see that increasing computer speed by 1000 times, **Algorithm A** increases the size of problems solvable within a unit time by 10 times, while for **Algorithm B**, the size only increases by 9.97 (~10).)

编写:蒙应杰

图 2.3 启动图形化安装界面

对于时间复杂度，会关注时间复杂度的 最坏情况和时间复杂度的平均情况。

2.3.2 复杂性分析

不需要知道精确的数值，只需要知道他一个规模

- 时间复杂性的分析

1. 根据问题的特点合理选择一种或几种操作作为整个算法的“标准操作”(假设循环执行次数最多, 循环就是标准操作)
2. 确定每个算法在给定输入下总共执行了多少次标准操作, 并根据次数推导求出时间函数
3. 确定该函数的阶

- 空间复杂性的分析

1. 根据问题的特点合理选择一种或几种操作作为整个算法的“标准操作”(假设循环执行次数最多, 循环就是标准操作)
2. 确定每个算法在给定输入下总共执行了多少次标准操作, 并根据次数推导求出时间函数
3. 确定该函数的阶

要求各个算法的存储结构一样的前提下, 关注算法所需的附加存储空间复杂性

2.4 算法设计方法概论

2.4.1 概述

- 用科学的方法进行算法设计
- 最常用方法：自顶向下逐步求精

顶层：问题的总和、抽象全貌

底层：问题的具体化、展开、细节

求精的方法：

1. 分而治之

将问题划分为一些不相交的部分，依次解决

2. 作出有限进展

采用一个朝解的方向得到有限进展的方法，反复应用，逐渐逼近

3. 情况分析

对问题各种情况给予分析，选择适当方案。

- 健壮性

- 高效性

2.4.2 算法设计的基本技术

以下内容不做要求

- 穷举法
- 分治法
- 回溯法
- 分支界限法
- 动态规划法
- 贪心法

2.5 算法描述语言

2.5.1 PDL 概述

Program、Design、Language

即伪码语言，主要用来书写软件设计的规约，是基于我们自然语言与具体的成设计语言之间的一种语言。这是一种保留计算机、程序设计、语言的基本框架和描述形式，并去掉一些特异性和直接性的要求，再结合自然语言所形成的一种用于描述算法处理的逻辑语言。

2.5.2 PDL 的优势

- 表达能力强，具有关键字的固定语法。提供了特定的结构化控制结构
- 引入了自然语言的一些习惯，结构比较清晰，简单易读
- 容易转化为任何一种程序设计语言代码（可由 PDL 生成程序代码）

2.5.3 PDL 书写及要求

1. 算法的框架

- 一般过程的书写框架

```
1 PROC 过程名 (I/O参数) ;  
2 BEGIN  
3     语句组  
4 END;
```

- 函数过程的书写框架

```
1 FUNC 函数名 (I/O参数) : 类型名 ;  
2 BEGIN  
3     语句组  
4 END;
```

2. 词的定义及说明

标识符：按照一定的规则形成的具有特定含义的一个词

- 过程名：调用前需定义
- 常量名、变量名：使用前需说明

例如 VAR i,j,k:integer

常见的数据类型名写法及表示：

- 整数型：integer
- 实数型：real
- 布尔型：boolean
- 字符型（单字符）：char
- 子介型（用于表达范围）：下界..上界，例如 40..90(表示 40-90), 'A'..'G'
- 枚举类型：0,1,2,3 元素次序不能变
- 构造类型
 - * 数组型：

1 ARRAY[下标类型] OF 成分类型

例如：

1 A: ARRAY[1..20] OF integer

1 B: ARRAY[1..20, -10..20] OF real // (B是点集,
二维数组)

* 记录型：

1 RECORD

2 域标识符1：类型1

3 ...

4 域标识符n：类型n

5 END

例如：

1 A=RECORD

2 Name:ARRAY[1…8] OF char;

3 Sex:0..1

4 Age:integer;

5 END

- 指针类型：

1 类型名

例如：

1 TYPE A=integer; //指针类型

1 VAR B: integer; //指针变量

3. 基本语序

- 赋值语句：

1 变量名 表达式

- 流程图

- 条件语句

- 形式一

1 if 条件 then 语句组

- 形式二

```
1 if 条件 then 语句组1  
2           else 语句组2
```

- 循环语句

- 当型 (while)

```
1 WHILE 条件 DO  
2           语句组；
```

- 直到型 (repeat)

```
1 REPEAT  
2           语句组；  
3 UNTIL 条件
```

- 从到型 (for-to)

- * 默认步长为 1:

```
1 FOR 变量 初值 TO 终值 DO  
2           语句组；
```

- * 自定义步长:

```
1 FOR 变量 初值 TO 终值 STEP 步长值 DO  
2           语句组；
```

- * 倒数:

```
1 FOR 变量 初值 DOWNTO 终值 DO  
2           语句组；
```

- 输入语句:

```
1 read(变量名表);
```

例:

```
1 read(x,y,z);
```

- 输出语句

```
1 write(变量名表);
```

4. 拓展语序

- 情况语句

```
1 CASE  
2     条件 1: 语句组 1;  
3     条件 2: 语句组 2;  
4     .....  
5     条件 n: 语句组 n;  
6     [ELSE 语句组 n+1]  
7 ENDCASE
```

- 一般过程调用语句:

```
1 Call 过程名;
```

- 函数过程调用: 通过在表达式中引用函数名完成, 即被引用函数名出现在表达式中。
- 出错提示语句:

```
1 error(错误信息);
```

- 终结语句

```
1 Exit \\ 算法转向正常结束
```

```
1 Return \\ 算法转向正常结束, 携带值离开
```

```
1 Abort \\ 中途废止 (中止)
```

- 复合语句

```
1 [ 简单语句 1;  
2     简单语句 2;  
3     .....  
4     简单语句 n; ]
```

或用 Begin End 代替括号

- 动态符号

– 储存单元的引用:

```
1 指针变量名
```

例:

```
1 x
```

– 动态空间分配:

1 New(P)

– 动态空间回收:

1 Dispose(P)

– 空地址的表示:

1 Nil

第三章 线性表

3.1 线性表及其运算

3.1.1 线性表的定义

线性表的定义：一个线性表是 n_0 个数据元素 a_1, a_2, \dots, a_n 的有限序列，序列中除第一及最后一个元素以外，每个元素有且只有一个直接前驱和直接后继。

简称表，可表示为： $A = (a_1, a_2, \dots, a_n)$

```
1 ai:datatype //表示  $a_i$  项可以是任何类型
```

3.1.2 线性表的特征

- 有限的。线性表的表长：线性表元素的个数。控标的长度定义为 0
- 元素呈线性关系。元素的位置只取决于他们自己的逻辑顺序。

3.1.3 线性表的运算

- 确定线性表的长度 n
- 存取线性表的第 i 个数据元素，检验或改变某个数据项的值
- 在第 $i-1$ 个和第 i 个数据元素之间插入一个新的数据元素。约定插入的元素是第 i 个元素的直接前驱
- 删去第 i 个元素
- 将两个或两个以上的线性表合并成一个线性表
- 将一个线性表拆分成两个或两个以上的线性表
- 重新复制一个线性表
- 对线性表中的数据元素依据某一种规则进行重组

3.2 线性表的储存表示

- 线性表的向量表示
 - 存储方法顺序地分配存储单元，且每个数据元素占据相同大小的存储空间（顺序且等长）
 - 数据访问 TODO
 - 向量存储结构特性

- * 储存分配呈线性结构
- * 属于随机存储结构（访问一个元素的代价与元素位置无关，即访问任何一个元素（找地址）的运算量一样，一维数组，通过下标变量来访问（数组构造的本质即算公式））
- 向量存储结构的形式化表示可用一个一维数组表示，由于数组属于静态结构，其空间规模须事先定义（1..max），要有个计数器记录数组长度（空间规模）

表述形式：

```

1 TYPE SQLIST:ARRAY [1..max] OF datatype;
          ## 内容
2 VAR n:0..max;
          ## 计数器（记录
          数组长度）

1 TYPE SQLIST= RECORD
2
          data:ARRAY
          [1..
          max
          ]
          OF
          datatype
          ;
          ## 内容 n:0..max; ## 计数器（记录数组长度）
4      END;

```

- 插入
- 删除
- 小结
- 线性表的链表表示
 - 单链表表示

```

1 TYPE pointer = node;
2           node=RECORD
3                   data: datatype;
4                   next: pointer;
5           END;
6           link=pointer;

```

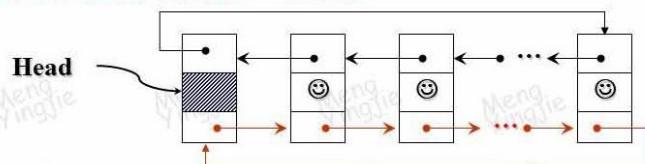
- 带表头的单链表表示
- 带表头结点的循环单链表表示
- 带表头结点的双向循环链表表示

```

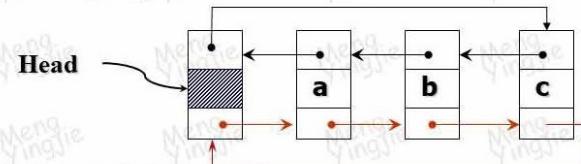
1 TYPE pointer = node;
2           node=RECORD
3                   Left: pointer;
4                   data: datatype;
5
6                   right: pointer;
7           END;
8           dblink=pointer;

```

带表头的双向循环链表的一般表示:



例1，线性表A=(a,b,c)的存储示意图：



例2，线性表A=() 的存储示意图：

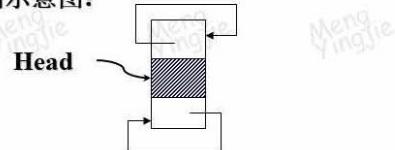
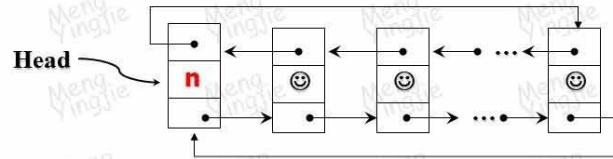


图 3.1 启动图形化安装界面

实际使用中双向链表的一些其它变化：

例1. 表头结点的数据项的利用：



例2. 指针数据项的变化：

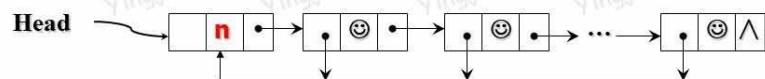


图 3.2 启动图形化安装界面

- 线性表的应用

第四章 栈和队列

4.1 栈及其运算

栈是工具性数据结构

4.1.1 绪论

4.1.2 栈的基本定义

栈是一个下限为常数，上限可变化的向量（或者反之）。有时称为堆栈或堆阵。

后进先出表（LIFO）

可变化一端称为栈顶，不变化的一端称为栈底

4.1.3 与线性表的关系

- 相同点

- 逻辑关系都是线性关系

- 不同点

- 线性表可以从任意位置增删，栈的增删只能从表尾进行

- 栈的运算是线性表的一个子集，且这个子集还需加以约束

4.1.4 栈的运算

- 入栈，PUSH(S)，完成往栈中加入元素的过程，即插入操作，也称为压栈
- 出栈，POP(S)，完成从栈中取出元素的过程，即删除操作，也称为弹出

4.1.5 栈的存储与运算的实现

4.1.6 多栈共存问题

4.2 栈的应用

4.3 队列及其运算

- 绪论分时和并行，主要处理对稀缺资源的争夺
- 队列的定义队列是一个下限和上限只能增加而不能减少（下限和上限的指针只能往一个方向移动）的向量（或者反之）

先进先出表 (FIFO)

- 队列与线性表
 - 相同点
 - * 逻辑关系都是线性关系
 - 不同点
 - * 线性表可以从任意位置增删，队列的只能一端插入一端删除
 - * 队列的运算是线性表的一个子集，且这个子集还需加以约束
- 队列的运算
 - 出队
 - 入队
- 队列的存储与运算的实现

4.4 受限的栈及队列（了解）

- 双端队列

双端队列是一种所有的插入和删除都限制在表的两端进行的线性表

- 双栈

双栈是一种加限制的双端队列，即从哪端进就只能从哪端出，就像是两个底部相连的栈

- 超队列

超队列是一种删除受限制的双端队列，删除限制在一端，插入可以在两端

- 超栈

超栈是一种插入受限制的双端队列，插入限制在一端，删除可以在两端

第五章 串

5.1 串及其运算

- 概述

字符串是线性表模型数据元素实例化的体现

全称字符串

早期：作为输入输出的常量和提示

热点：中文信息处理

- 串的定义

一个由零个或多个字符组成的有穷序列称为串

简记为 $A = 'a_1 \ a_2 \ a_3 \dots \ a_n'$

串的长度：串中所含的字符个数

空串：串长为零的串，记作

非空串

空白串：空白字符组成的串，记作“”

串的相等：串长也相等，对应位置上的字符也一样

字串/主串：一个串中任意个连续字符组成的子序列称为该串的子串，该串成为它的所有子串的主串。不一定是一个真子集

- 串的运算

- 赋值

`assign(S,chars)` 将字符串常量 `chars` 赋给字符串变量 `S`

- 连接

`concatenation(S,T)` 将字符串 `S` 和 `T` 联接在一起

- 取子串

`substring(S,m,n)` 从第 `m` 个位置开始取 `n` 个连续字符（通常）/从第 `m` 个位置开始到第 `n` 个位置

- 求子串序号

`strindex(S,T,i)` 确定串 `T` 在 `S` 中第一次出现的位置 `i`

- 串的插入

`strinsert(S,T,i)`

- 串的删除
 - strdelete(S,m,n)
- 串的复制
 - copy(S,T)
- 串的置换
 - replace(A,B,C) (找到 A 中的 B 用 C 来替换)
 - replace(S,m,n,T) (把 S 字符串的 (m 到 n/从 m 开始的 n 个字符) 用 T 来替换)
- 串的存储
 - 顺序储存
 - 一般用向量来表示，通常称为字符串数组、字符串变量、字符串常量
 - * 压缩模式
 - 按字节存储数据 (C 语言)
 - * 非压缩格式
 - 运算器以字为单位运算，存储器以字节为单位，为了避免转换浪费的时间，存储器以字为单位存储一个字符，变成非压缩模式（只存一个字母需要一个字的空间，比较浪费）
 - 索引储存
 - 多用在对于多个字符串常量或者变量的组织
 - 链接储存
 - 理论上研究的多，实际上用的少，不易于实现

5.2 串的模式匹配

- 概述

在模式分类或者问题回答系统等方面，将输入模式与样本模式进行匹配的过程啊，我们就把它称为模式匹配

通常把 S 称为目标 (或正文)，把 P 称为模式，把从目标 S 中查找模式 P 的过程称为模式匹配

串的匹配是模式匹配的实例化
- 匹配的朴素算法 (Brute-Force 算法)

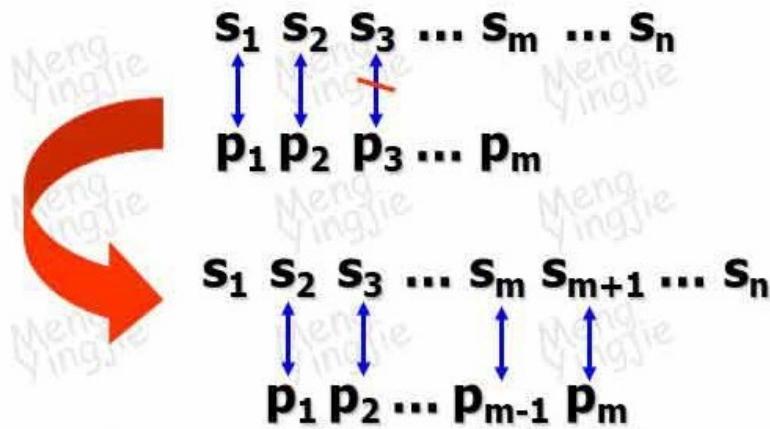


图 5.1 启动图形化安装界面

处理基本思想：对正文顺序搜索

1. 逐次比较（对应字符依次比较）
2. 发现不匹配时，将 P 相对于 S 右移一位
3. 重复上述过程，直到成功或扫描完 S

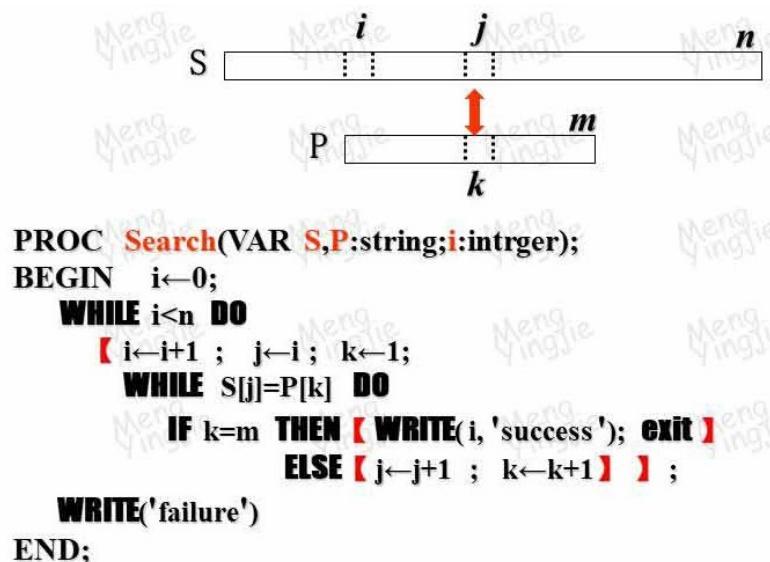


图 5.2 启动图形化安装界面

优化算法：KMP 算法

5.3 例题

2019 年兰州大学开源社区纳新面试题中有一题关于字符串的运算，现将我的解决方案附于后文。

给出一个文本文件，其中每个单词不包括空格及跨行，单词由字符序列构成且不区分大小写，完成以下功能：统计给定单词在文本文件中出现的总次数。

注：允许使用 string.h

```
1 #include<stdio.h>
2 #include<string.h>
3
4 char word[100];
5 char text[10000];
6 int count = 0;
7
8 char trans(char c){
9     if(c >= 'A' && c <= 'Z'){
10         c = c + 'a' - 'A';
11     }
12     return c;
13 }
14
15 void getword() {
16     char c;
17     int i = 1;
18     while (1) {
19         c = getchar();
20         if (c == '\n') break;
21         word[i] = trans(c);
22         i++;
23     }
24     word[0] = i - 1; //第0位存放单词长度
25     return;
26 }
27
28 void gettext() {
29     char c;
30     int i = 1;
31     while (1) {
```

```
32         c = getchar();
33         if (c == EOF) break;
34         text[i] = trans(c);
35         i++;
36     }
37     text[0] = i - 1; //第 0 位存放文本长度
38     return ;
39 }
40
41 void check() {
42
43     int i = 0, j = 0;
44     int flag = 1;
45     while (1) {
46         i++;
47         j++;
48         if (i == text[0]) break;
49         if (text[i] == '\n' || text[i] == ' ') {
50             j = 0;
51             flag = 1;
52             continue;
53         }
54         if (flag == 0) {
55             continue;
56         }
57         if (text[i] != word[j])
58             flag = 0;
59         if (j == word[0]) {
60             if (flag == 1) {
61                 if (text[i + 1] == '\n' || text[i + 1] == ' ')
62                     count++;
63             }
64             flag = 1;
```

```
65         }
66     }
67 }
68
69 int main()
70 {
71     freopen( "C:\\\\Users\\\\Kente\\\\Desktop\\\\123\\\\text.txt" , "r"
72         , stdin); //假设给出的文本文件第一排是给定单词，第二
73         //排及之后是要统计的文本
74     getword();
75     gettext();
76     check();
77     printf( "%d" , count);
78     return 0;
79 }
80
81 /*text.txt
82 ABC
83 Abc abC dasasdsa abc
84 dasdasdasdasd dasdasdasdasd
85 abc ab
86 */
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
788
789
789
790
791
792
793
794
795
796
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
887
888
889
889
890
891
892
893
894
895
896
897
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
918
919
919
920
921
922
923
924
925
926
927
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
948
949
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
978
979
979
980
981
982
983
984
985
986
987
988
989
989
990
991
992
993
994
995
996
997
998
999
999
1000
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1097
1098
1099
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1188
1189
1190
1191
1192
1193
1194
1195
1196
1196
1197
1198
1199
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1297
1298
1299
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1397
1398
1399
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1497
1498
1499
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1597
1598
1599
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1697
1698
1699
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1797
1798
1799
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1897
1898
1899
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1997
1998
1999
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2097
2098
2099
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2197
2198
2199
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2278
2279
2280
2281
```

第六章 数组和广义表

第七章 树形结构

7.1 树的基本定义和运算

7.1.1 树形结构的概述

- 与线性结构的差异
直接后继可以是多个
- 树形结构的地位
树是计算机中最重要的非线性结构
- 讨论内容
 - 逻辑角度
 - 存储方式
 - 应用
 - 模型
 - * 树
 - * 二叉树

7.1.2 树的基本定义

树 T , 是满足如下性质的有限个节点组成的非空集合

- T 中有且仅有一个称为根的结点
- 除根结点之外, 其余结点分成 $m(m>0)$ 个不相交的集合 T_1, T_2, \dots, T_m , 其中每个 T_i 都是树, 而且都称为 T 的子树。

注意

- 是递归定义
- 定义中对于子树的个数及次序没有进行任何约束
- 与图论中的树不同, 是有向图中的根树的特例。隐含了向下的方向性

7.1.3 有序树的基本定义

在树 T 中如果子树 T_1, T_2, \dots, T_n 的相对次序是重要的 (即有序的), 则称 T 为有向有序树, 简称有序树。

默认树是有序树

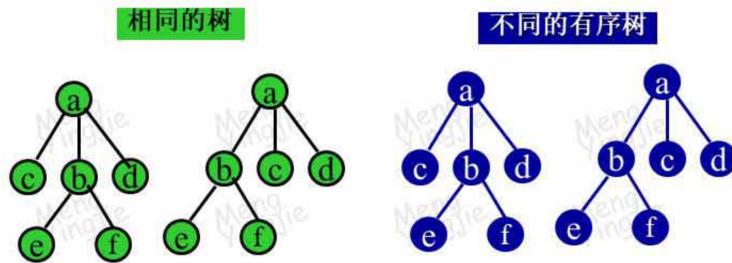


图 7.1 启动图形化安装界面

7.1.4 森林（树林）

森林是零棵或多棵不相交的树的集合（通常是有序集合）

7.2 二叉树

- 定义
- 特性
- 二叉树的存储表示

7.3 遍历二叉树

7.3.1 概念

定义：对于给定的数据结构，系统地访问该结构中的每个结点，且每个结点仅被访问一次的操作过程称遍历

系统地：按照一定的规律、次序

访问：对元素进行的某种操作

遍历本质上是一个运算过程（序列）

对于给定的二叉树，系统地访问该结构中的每个结点，且每个结点仅被访问一次的操作过程称遍历

7.3.2 二叉树的遍历次序（order）

- 层次策略
 - 自上而下
 1. 从左到右
 2. 从右到左
 - 自下而上

- 深度策略遍历针对结构，是一个过程；访问针对一个点，是一个动作。

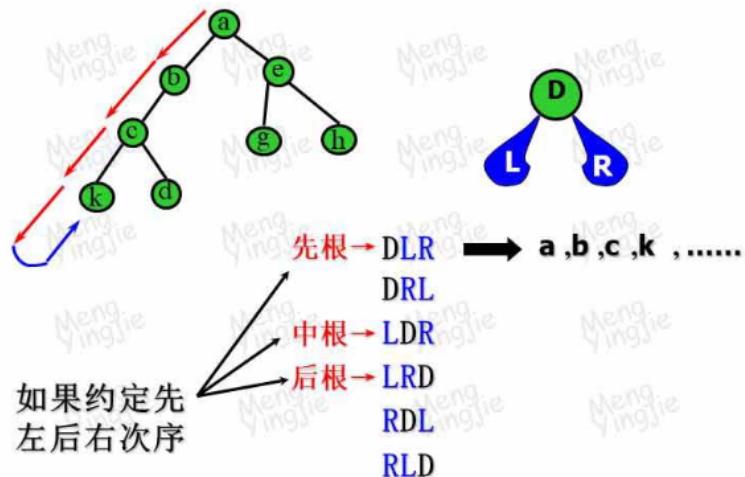


图 7.2 启动图形化安装界面

- 先根（前序） DLR

先拿到根，然后从左子树走，又是二叉树，则访问了左子树的根节点，再往下走，直到访问到 k，左子树为空，回到 c，访问右子树。沿着纵深往下搜索，只要有子树就往下搜索，先不管兄弟。

- 中根 LDR

- 后根（后序） LRD

7.4 树、森林与二叉树的转换

7.4.1 概述

转换原因

- 二叉树有许多优良特性，而树没有
- n 叉树中二叉树的空间利用率最高

7.4.2 树转为二叉树

这里约定树为有序树，子树的次序相对固定

转换规则：

- 加线（亲兄弟之间加上虚连线）
- 抹线（除了最左子节点之外，抹掉所有节点与其余子节点的连线）
- 调整（调成二叉树的样子，原有连线在左，新加连线在右）

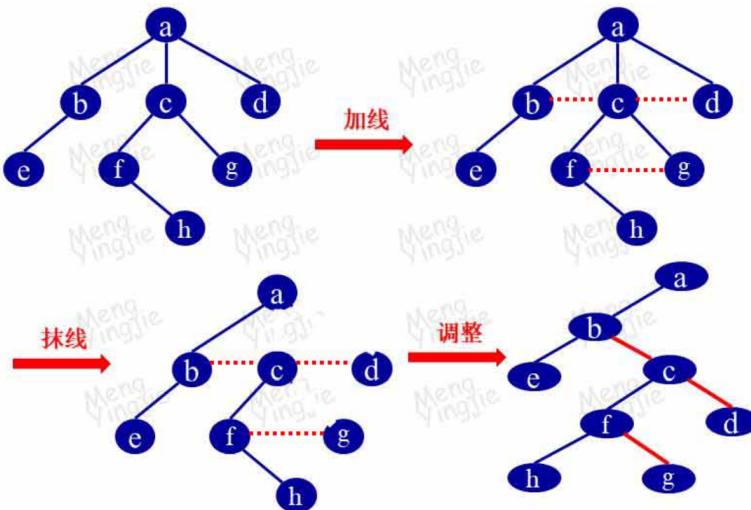


图 7.3 启动图形化安装界面

特点：

- 原有兄弟成为右结点及右结点的右列结点
- 根子树仅有左节点

7.4.3 二叉树还原为树

转换规则：

1. 加线（如果有节点 X 是父结点的左子树，那么它的右子树，右子树的右子树... 都与 X 的父节点加线）
2. 抹线（所有节点抹掉与右子树的连线）
3. 调整（让节点按层次分布）

7.4.4 森林转为二叉树

两种方法：

- 方法一：树转换
将每棵树转为二叉树
- 方法二：二叉树连接

依据转换得到的二叉树的次序，将后一棵作为前一棵根节点的右子树，因为树转二叉树的根节点无右子树

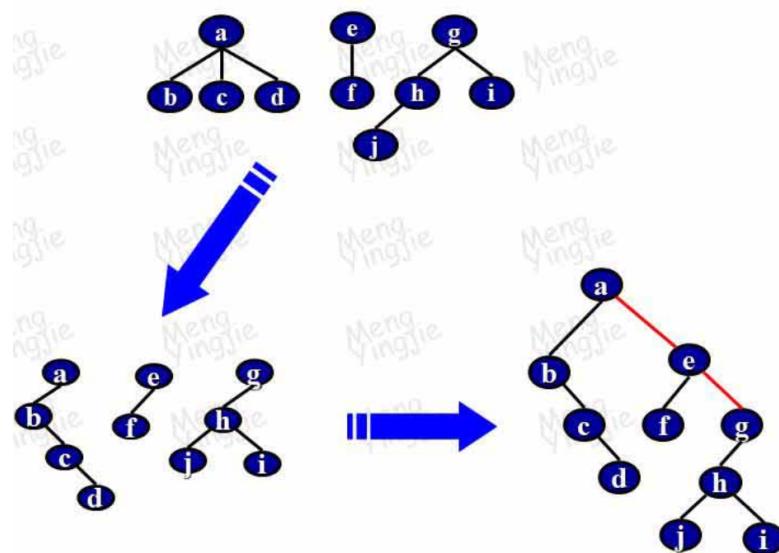


图 7.4 启动图形化安装界面

7.4.5 二叉树转为森林

转换规则：

1. 抹线（沿着根节点，抹掉其右子树与根节点的连线，不断往右下搜索）
2. 还原（把每棵二叉树还原树）

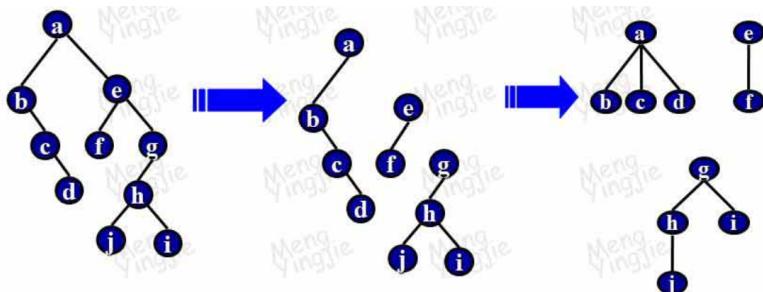


图 7.5 启动图形化安装界面

7.4.6 树的遍历

- 方法一：树转换

将每棵树转为二叉树后再处理，处理完成后还原

- 方法二：直接处理

遍历次序

– 层次策略

* 自上而下

1. 从左到右
 2. 从右到左
- * 自下而上
- 深度策略
- * 先根（前序）DLR
 1. 访问树的根节点
 2. 先根次序下依次遍历树的根节点的每个子树
 - * 中根 LDR
一般不讨论
 - * 后根（后序）LRD
 1. 后根次序下依次遍历树的根节点的每个子树
 2. 访问树的根节点

7.4.7 森林的遍历

- 方法一：树转换

将每棵树转为二叉树后再处理，处理完成后还原

- 方法二：直接处理

遍历次序

- 层次策略

* 自上而下

1. 从左到右

2. 从右到左

* 自下而上

- 深度策略

* 先根（前序）DLR

1. 访问第一棵树的根节点

2. 先根次序下依次遍历第一棵树的根节点的每个子树

3. 先根次序下依次遍历其余的树

* 中根 LDR

一般不讨论

* 后根（后序）LRD

1. 后根次序下依次遍历第一棵树的根节点的每个子树
2. 访问第一棵树的根节点
3. 后根次序下依次遍历其余的树

7.5 线索树

7.6 树形结构的应用

第八章 图结构

8.1 基本概念

8.1.1 概述

与线性结构和树形结构的差异

- 一个元素可以有多个直接前驱和后继
- 是多对多的数据关系

地位

图结构非常重要的非线性结构。

讨论内容

- 计算机中的图的表示方法
- 图的遍历
- 求生成树
- 找最短路径
- 拓扑排序
- 求关键路径

8.1.2 概念

图

由 $n(n_1)$ 个结点 v_1, v_2, \dots, v_n 构成的数据 G 称为图，若结点集 $V = v_1, v_2, \dots, v_n$ 上定义的称为后继的关系 E 是非自反的。

可表示为 $G = (V, E)$, V 称为顶点集, E 称为边集。只相当于研究的图相当于离散数学中的简单图，相当于简单图，即不包含重边和环（自回路），不研究：

- 带自身环的图
- 多重图

有向图

在图 G 中，若每个关系都是顶点的有序对，则称 G 为有向图。

有向图的表示有两种方法：

- 形式化（符号）表示
用有向线段指明了次序（方向）。
- 图形描述
一般用尖括弧表示顶点的有序对。

图G:

$$\begin{aligned} V(G) &= \{v_1, v_2, v_3\}, \\ E(G) &= \{\langle v_1, v_2 \rangle, \langle v_2, v_1 \rangle, \langle v_2, v_3 \rangle\} \end{aligned}$$

(a) 形式化（符号）表示

(b) 图形描述

图 8.1 图的表示方法

在有向图中，若 $\langle v_i, v_j \rangle \in E(G)$

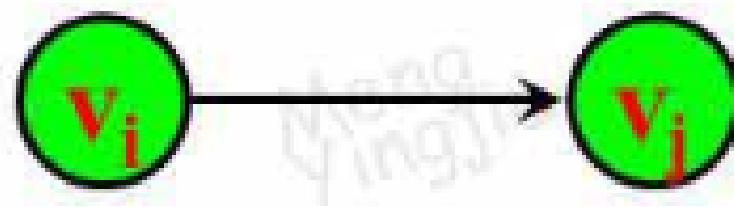


图 8.2 启动图形化安装界面

则称 v_i 是边的始点（或尾）； v_j 是边的终点（或头）；并称 v_i 邻接到 v_j ， v_j 是从 v_i 邻接过来的；称边 $\langle v_i, v_j \rangle$ 关联（或依附）于顶点 v_i 和 v_j 。

推论：在 n 个结点的有向图中，其最大边数为 $n(n-1)$ 。把等于 $n(n-1)$ 条边的图称有向完全图。

无向图

在图 G 中, 如果每个关系都是顶点的无序对, 则称 G 为无向图。有相图的表示有两种方法:

- 形式化(符号)表示

用无向线段表示(方向)。

- 图形描述

一般用圆括弧表示顶点的无序对

在无向图中, 若 $(v_i, v_j) \in E(G)$

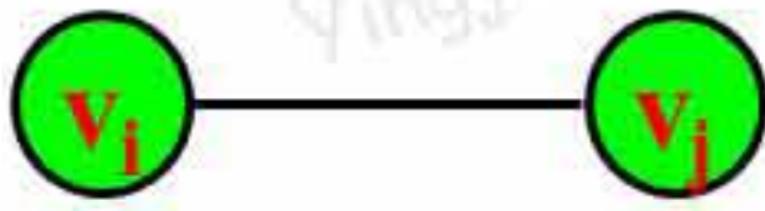


图 8.3 启动图形化安装界面

则称 v_i 和 v_j 是邻接的; 并称边 (v_i, v_j) 关联(或依附)于顶点 v_i 和 v_j .

在无向图中 (v_1, v_2) 和 (v_2, v_1) 这两个结点偶对表示同一条边。

推论: 在 n 个结点的无向图中, 其最大边数为 $n(n-1)/2$ 。把等于 $n(n-1)/2$ 条边的图称无向完全图。

顶点的度、入度、出度

在图中, 顶点的度(degree), 就是与该顶点关联的边的数目。

若 G 为无向图, 则度为无向图中与某元素有关的元素的数量。

若 G 为有向图, 则

- 把以 v_i 为终点(头)的边的数目称作 v_i 的入度(incoming degree,in-degree)(即指向该元素的边数)
- 把以 v_i 为始点(尾)的边的数目称作 v_i 的出度(out going degree)(即从该元素指出去的边数)

在有向图中, 出度为 0 的顶点称终端顶点(叶子)。

推论: 设图 G 有 n 个结点, t 条边, 若 d_i 为顶点 v_i 的度, 显然有: TODO

子图

设图 $G = (V, E)$, 如果有图 $G' = (V', E')$, 且 E' 包含于 E , V' 包含于 V , 则称 G' 为 G 的子图。

如果图 G 的子图包含 G 的所有顶点, 则该子图称为 G 的生成子图。

自己可以是自己的子图 v 。

路径、回路、图根

对于图 $G=(V,E)$, 若存在结点序列 $v_p, v_{i1}, v_{i2}, \dots, v_{im}, v_q$ 使得 $(v_p, v_{i1}), (v_{i1}, v_{i2}), \dots, (v_{im}, v_q)$ 都在 $E(G)$ 中 (对于有向图使得 $\langle v_p, v_{i1} \rangle, \langle v_{i1}, v_{i2} \rangle, \dots, \langle v_{im}, v_q \rangle$ 都在 $E(G)$ 中), 则称从顶点 v_p 到 v_q 存在一条路径 (或通路, path), 路径长度定义为这条路径上边的数目。(路径: 一个特定的点边交替序列)。

v_p 到 v_q 的路径可以简写为: $v_p, v_{i1}, v_{i2}, \dots, v_{im}, v_q$ (用结点序简写)。

如果一条路径上的顶点除 v_p 和 v_q 可以相同以外其它顶点都不相同, 则此路径为一简单路径。(即中间没有重复经过的点)。

把 $v_p=v_q$ 的简单路径, 称作回路 (或环, loop)。(不包含自身环)。

若在一个有向图中存在一个顶点 v_0 , 从此顶点有路径可以到达图中其它所有顶点, 则此有向图称为是有根的图, v_0 称作图根。(树是有向图中的有根图的特殊情况)

连通性

- 无向图

两个顶点的连通: 在无向图 G 中, 顶点 u 和 $v(uv)$ 之间存在一条路径, 则称顶点 u 和顶点 v 是连通的。

图的连通: 对于无向图 G 中的任意两个顶点 $u, v(uv)$ 都是连通的则称此无向图是连通的, 或称 G 为连通图。(仅有无向图才谈图的连通)。

最大 (或极大) 连通子图: 无向图的连通分量。连通分量是一个 (或多个) 子图, 不是一个数字。

- 有向图

两个顶点的连通: 在有向图 G 中, 顶点 u 和 $v(uv)$ 之间存在一条从 u 到 v 和 v 到 u 的 (有向) 路径, 则称顶点 u 和顶点 v 是连通的。(就是两个点能互相到达)

图的强连通: 有向图 G 中的任意两个顶点 $u, v(uv)$ 都是连通的则称此有向图是强连通的, 或称 G 为强连通图。

一个有向图的强连通分量定义为该图的最大 (或极大) 强连通子图 (的个数)。(有向图不谈图的连通, 只谈图的强/弱连通)

网

给图的每一条边加一个 (非负的) 数, 这个与图的边相关的数值称为权 (weight)。带权的图称为网。带权的连通图称为网络。(网络针对无向图)

8.2 图的存储

8.2.1 邻接矩阵

定义：表示顶点之间邻接关系的矩阵称邻接矩阵

图非网时

用 boolean 矩阵表示

设图有 n_1 个顶点，则图 G 的邻接矩阵定义为 n 阶方阵 M，M 满足下列性质：

$$m_{ij} = \begin{cases} 1, & \text{若 } (v_i, v_j) \text{ 或 } \langle v_i, v_j \rangle \in E(G) \\ 0, & \text{反之} \end{cases}$$

图 8.4 启动图形化安装界面

对于无向图，它的邻接矩阵一定是一个对称矩阵，且矩阵第 i 行之和是顶点 v_i 的度

对于有向图，它的邻接矩阵一定是一个非对称矩阵，矩阵第 i 行之和是顶点 v_i 的出度，矩阵第 i 列之和是顶点 v_i 的入度。

网的邻接矩阵

把原来的存 1 改成存权值。

设图有 n_1 个顶点，则图 G 的邻接矩阵定义为 n 阶方阵 M，M 满足下列性质：

$$m_{ij} = \begin{cases} w_{ij}, & \text{若 } (v_i, v_j) \text{ 或 } \langle v_i, v_j \rangle \in E(G) \\ 0 \text{ 或 } \infty, & \text{反之} \end{cases}$$

图 8.5 启动图形化安装界面

在计算机中，可以取一个相对于问题而言相对大的数即可代表。

8.2.2 邻接表

对图的每个顶点建立一个链表，也称邻接链表表示。

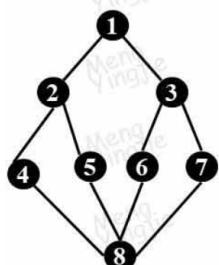
有 n 个结点时有 n 个链表，第 i 个链表中的结点：

- 在无向图中：是与 v_i 邻接的所有结点的收集。
- 在有向图中：是以 v_i 为始点的所有终点的收集。

无向图

二. 邻接表(Adjacency List)

举例：无向图



第 i 个链表中的结点个数：
总表结点数目：



图 8.6 启动图形化安装界面

开头的结点是线性结构，可以用向量或者链表存，此处用向量存开头的节点。

总链表结点数目： $2e$ （每条边出现了两次）

加上表头，存储空间需 $n+2e$

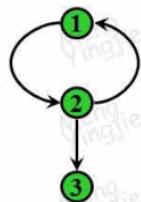
第 i 个链表中的结点个数： v_i 的度。

无向图的邻接表，链表中的顺序可以随意变换，不分先后（即同一个图也可以有不同的邻接表）

结点间是否有边判别比邻接矩阵困难。

有向图

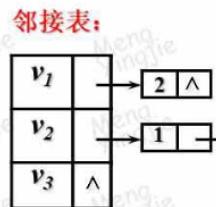
举例：有向图



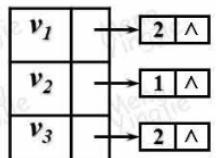
第*i*个链表中的结点个数

总表结点数目：

存储示意图：



逆邻接表：



对于网的邻接表，表结点可以再增加一个数据项即可。

图 8.7 启动图形化安装界面

表节点数目：e

总空间：n+e

第*i*个链表中的结点个数： v_i 的出度。

逆邻接表：以 v_i 为终点的所有始点的收集。可用于获取邻接过来的结点。

8.2.3 邻接多重表

邻接多重表以边为存储单位。

边 (u, v) 结点结构：



图 8.8 启动图形化安装界面

- mark：标志域，标记该边是否被处理，也可存放权值等信息
- u, v：关联边的两个顶点域
- ulink：下一条与依附于 u 的边的地址（指针）
- viink：下一条与依附于 v 的边的地址（指针）
- info：其他信息

8.3 图的遍历

8.3.1 概述

定义

给出图 G 和其中的任意一个顶点 v_0 (人为给出), 从 v_0 出发系统地访问 G 中所有的顶点, 且每个顶点仅被访问一次, 这一过程称为图的遍历 (graph traversal), 或遍历图。

辅助结构

须设立辅助结构, 可用数组 $\text{visited}[1..n]$ 表征

$$\text{visited}[v] = \begin{cases} 1, & v \text{已经被访问} \\ 0, & v \text{未被访问} \end{cases}$$

图 8.9 启动图形化安装界面

搜索策略

- 深度优先搜索 (DFS, depth-first search)
搜索中, 结点扩展的次序向某一个分支纵深推进, 到底后再回溯。
- 广度优先搜索 (BFS, breadth-first search)
搜索中, 对所在层次的所有结点逐个依次进行扩展后, 再推进到下一个层次进行扩展。

8.3.2 图的深度优先遍历

图的深度优先遍历的基本思想

1. 访问出发节点 v_0 , 标记 v_0 为已访问
2. 选择一个 v_0 邻接到的且未被访问过的结点 u
3. 从 u 开始进行深搜 (递归开始)

基于邻接表

面授部分讲解

8.3.3 图的广度优先遍历

图的广度优先遍历的基本思想

1. 访问出发点 v_0
2. 访问 v_0 邻接到的所有未被访问过的节点 v_1, v_2, \dots, v_t
3. 再依次访问 v_1, v_2, \dots, v_t 邻接到的所有未被访问的结点
4. 如此进行下去，直到无法找到未被访问的结点时，则本次搜索就算结束

8.4 连通性及最小生成树

8.4.1 连通分量和生成树

求图的连通分量

思想：将图的遍历中搜索算法 DFS(或 BFS) 中的访问动作具体化 (即输出顶点及与之关联的边)，每启动一次 DFS(或 BFS) 就可以得到 1 个连通分量。

求生成树

通过图的遍历可获取其生成子图。

对于连通图来说该生成子图，实际上是一棵树结构 (恰好有 $n-1$ 条边，它们把 n 个事物关联起来，可称其是原图的极小连通子图)

以下三种情况：

- 图是连通的
- 图是强连通的
- 出发点 v_0 为图根

在遍历过程中，访问的节点和经过的边，所构成的生成子图，恰好可以构成一个树形结构，称为生成树。对于有 n 个顶点的连通图，其生成树具有 $n-1$ 条边，即生成树是图的最小结构。

对于不连通的无向图和不是强连通的有向图，从任一结点出发只能得到生成森林。

生成树因为出发点不同，深、广不同而不唯一。

基于遍历方法我们可以构造极小连通子图 (一谈连通图一定是无向图) (即节点不变，关联关系最小)。

8.4.2 最小生成树

最小生成树是边的权值之和最小的网络的生成树

普里姆 (Prim) 算法

基本思想：设 $V=1,2,\dots,n$ 是图 G 的顶点集合，PRIM 算法由一个初值为 v_0 的集合 U 开始，它每次生成一条边，逐渐长成一棵具有最小代价的生成树。

算法在每一步中都找出一个最小权的边 (u,v) , 其中 $u \in U, v \in V-U$. (U, V 是点集, U 是已经找到的点的集合, V 是所有点)

即从任意一个点出发找到已找到的点集到未找到的点集中权值最小的一条路径不断连接即可

v_0 : 种子节点, 可任选

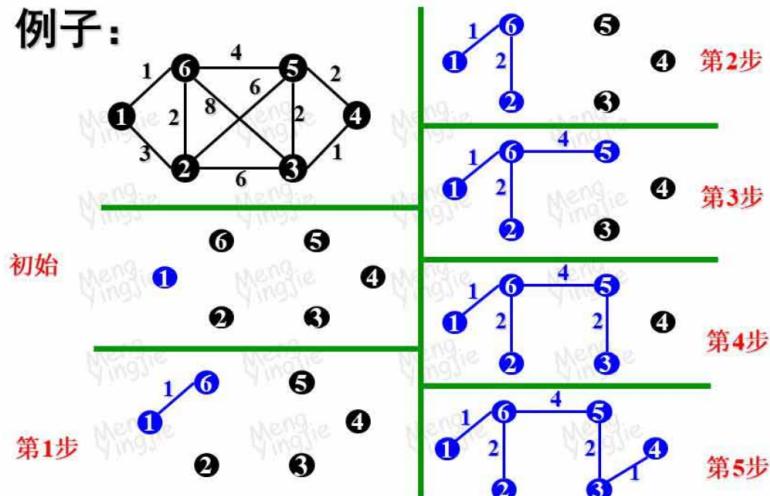


图 8.10 普里姆 (Prim) 算法例子

克鲁斯卡尔 (Kruskal) 算法

基本思想：对于图 $G=(V,E)$, 该算法初始化从 $T=(V, \emptyset)$ 开始, 此时生成树是由 n 个连通分量（即 n 个孤立顶点组成）组成的一个制图——目标使得 n 个连通分量逐渐成为一个连通分量。具体做法：

1. 初始化从 $T=(V, \emptyset)$ 开始, 即 $E=\emptyset$
2. 按照权值递增的顺序逐个考虑 E 中的每条边：
 - (a) 若该边连通了在两个不同连通分量中的顶点, 则将该边填加到 T 中
 - (b) 重复 (1), 一旦 T 中包含了 $n-1$ 条边, 则终止运算

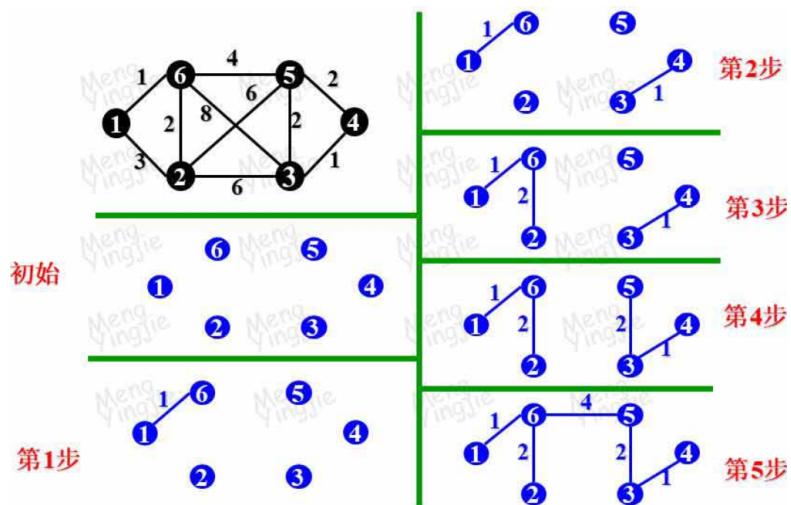


图 8.11 克鲁斯卡尔 (Kruskal) 例子

8.5 有向无环图及应用

8.6 最短路径

第九章 latex 部分用法简介

注意啊，看这个教程，template.pdf 配合 template.tex 一起看，才能学习 latex 怎么用的

网页跳转怎么用？图片插入怎么用？图片横着两个并排站呢？代码怎么插入？表格听说挺复杂？公式听说也挺难的

哈哈哈，你说你还不知道什么是 LaTeX，你去分不清 XeLaTeX、pdfLaTeX，百度一下竟然还让我安装 TexLive，这也就算了，甚至还有人说 vscode? sublime text3? texstudio? Texmaker? 我只是想写个论文排版方便一些，你要干嘛？

上面这些问题，后面都会一点点介绍

9.1 用 latex 需要安装什么

需要安装 texlive，外加一个 IDE

9.1.1 texlive 下载安装

最近可能出了 2020 了，可以用兰大的镜像下载应该在用校园网时快一些，额，你还是用清华的镜像吧，我刚才找了一下，兰大镜像这会儿竟然挂了。。。

下载地址¹：点下面的字跳转浏览器下载了，方便吧

- TexLive2019 Windows 版
- TexLive2019 Mac 版

上面的文件直接双击安装一路 next 就行，但是 texlive 这是个啥？

用过 python 吧，texlive 就相当于你下载的 python 安装包，但是你总不能在终端里写代码吧，一般用 pycharm，这个就是 IDE，所以你需要再安装一个 IDE。

为什么没 linux 版？用的人不多，真心不想给。。。其实安装文件就是 windows 的那个版本

linux 系统图形界面安装 texlive

1. 安装 perl 组件：sudo apt-get install perl-tk
2. 加载该 ISO 文件：sudo mount -o loop texlive2019.iso /mnt （换掉文件路径即可）²

¹这个地址会自动更新比如 2020 版出了以后你下载的就是 2020 了

²注意：使用该命令会出现错误提示，mount: /dev/loop1 is write-protected, mounting read-

3. 启动图形化安装界面: cd /mnt & sudo ./install-tl -gui

注意倒数第二项, 改成是, 创建符号链接, 下面那个图是网上随便找的, 都差不多

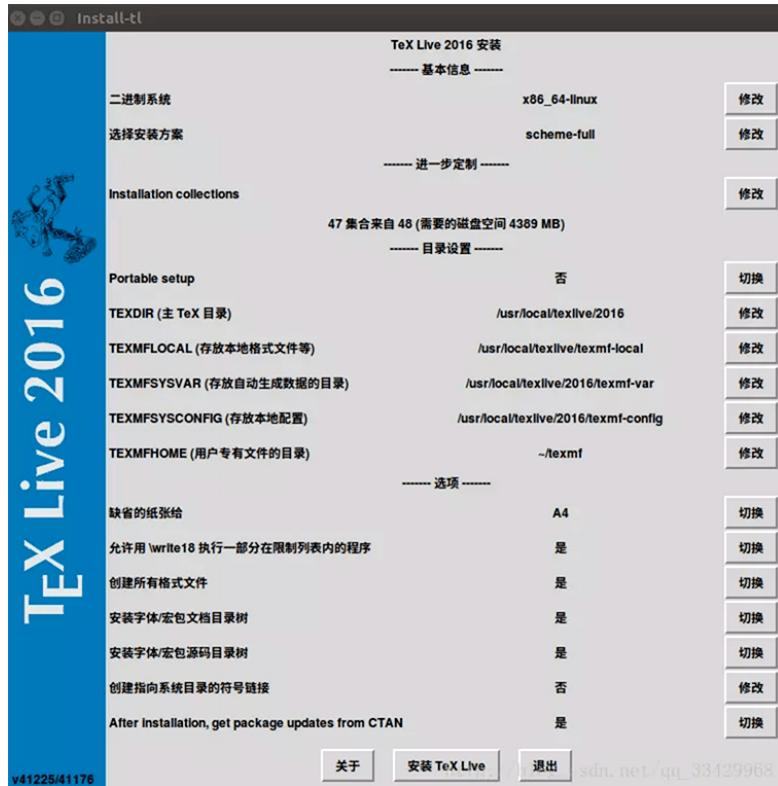


图 9.1 启动图形化安装界面

9.1.2 安装 IDE

在这之前, 请测试 texlive 是否安装成功!!! 在命令行输入 tex, 显示类似如下结果, 注意必须包含 “TeX Live 2020” 字样

```
1 This is TeX, Version 3.14159265 (TeX Live 2020) (
    preloaded format=tex)
2 **
```

如果确实安装了, 但是没有显示, 请根据各自系统自行百度配环境变量, 此处不再详细介绍

IDE 这个就是写论文的地方, 它会调用你刚才安装的 texlive, 具体用什么, 各有所爱, 我喜欢 sublime text3, 这个颜值是真的高, 而且体积小, 启动快, 可以预览公式什么的, 很多常用的代码可以自动提示补全, 但是这个需要安装插件

only. 不必管它

LaTexTools, pdf 需要安装其他的东西进行正反向跳转¹, 小白的话就算了吧, 想折腾, 百度一下吧

另外一个我比较喜欢的是 vscode, 也需要安装插件, 很方便, 和 sublime text3 差不多, 但是用起来简单一些, 也自己百度去吧

另外几个是 Texmaker、或者 texstudio², 这两个你点名字就跳转官网了, 这两个基本上是打开就可以用, 怎么用, 自己百度吧, 很多详细的图文教程

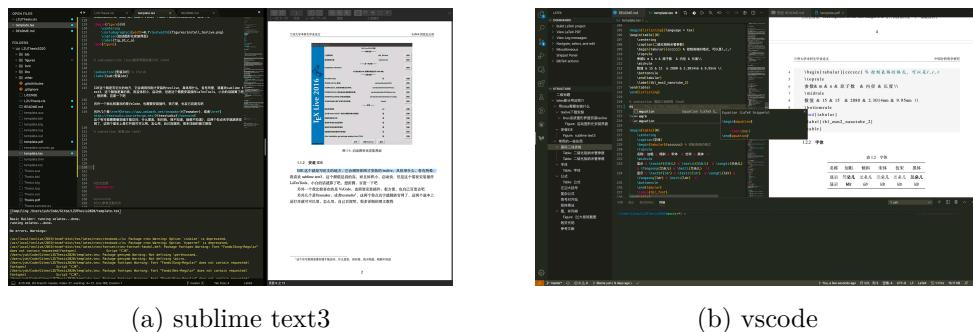


图 9.2 我用的 IDE

这两个 IDE 真的是特别好用, 不要用 TexMaker 或者 TexStudio 了, 连个自动提示都麻烦, 预览也没有, 最主要的是太丑了, 也不能换主题

9.2 常用的一些东西

用到相关的直接到这里复制, 然后修改就行

9.2.1 国际三线表格

表 9.1 二硫化钼纳米管参数

参数	m	n	换行一下	内径	长度
数值	15	15	2880	2.3014nm	9.95nm

¹就是点 tex 文件某一行跳转到 pdf 对应的地方, 点击 pdf 跳转到 tex 对应的那一行, mac 上安装 skim, windows 安装 sumatra

²这个有可能需要翻墙才能访问, 什么意思, 别问我, 我不知道, 啥都不知道

这个注意,有多少列,后面就要有多少个 c¹,这个 c 表示这一列居中 (center),靠左的话: l, 右: r;

那个 label 后面的名字自己取,但是不能有重复,是为了引用,比如这样,表 9.2, 方程、图片也是这样引用的,好处是,中间加一个表格导致这个表格的序号变了也没事,你不用再去修改其他地方的引用

```

1 \begin{table}[H]
2     \centering
3     \caption{二硫化钼纳米管参数}
4     \begin{tabular}{cccccc} \% 控制表格的格式, 可以是 l, c, r
5     \toprule
6     参数&m &n &原子数 &内径 &长度\\
7     \midrule
8     数值 & 15 & 15 & 2880 & 2.3014nm & 9.95nm \\
9     \bottomrule
10    \end{tabular}
11    \label{tbl_mos2_nanotube_2}
12 \end{table}
```

9.2.2 换页表格

我是真的没想到有的人表格居然这么长,竟然能有三页。。。

表 9.2 二硫化钼纳米管参数

参数	m	n	原子数	内径	长度
数值	15	15	2880	2.3014nm	9.95nm
数值 1	15	15	2880	2.3014nm	9.95nm
数值 2	15	15	2880	2.3014nm	9.95nm
数值 3	15	15	2880	2.3014nm	9.95nm
数值 4	15	15	2880	2.3014nm	9.95nm
数值 5	15	15	2880	2.3014nm	9.95nm
数值 6	15	15	2880	2.3014nm	9.95nm
数值 7	15	15	2880	2.3014nm	9.95nm
数值 8	15	15	2880	2.3014nm	9.95nm

¹否则会报错: Extra alignment tab has been changed to cr. 有什么报错百度一下一般就找到了

数值 9	15	15	2880	2.3014nm	9.95nm
数值 10	15	15	2880	2.3014nm	9.95nm
数值 11	15	15	2880	2.3014nm	9.95nm
数值 12	15	15	2880	2.3014nm	9.95nm
数值 13	15	15	2880	2.3014nm	9.95nm
数值 14	15	15	2880	2.3014nm	9.95nm
数值 15	15	15	2880	2.3014nm	9.95nm
数值 16	15	15	2880	2.3014nm	9.95nm
数值 17	15	15	2880	2.3014nm	9.95nm
数值 18	15	15	2880	2.3014nm	9.95nm
数值 19	15	15	2880	2.3014nm	9.95nm
数值 20	15	15	2880	2.3014nm	9.95nm

9.2.3 字体

表 9.3 字体

名称	加粗	倾斜	宋体	仿宋	黑体
显示	兰朵儿	兰朵儿	兰朵儿	兰朵儿	兰朵儿
显示	ldr	ldr	ldr	ldr	ldr

发现没，中文斜体没有效果的，你可以自定义，这个自己百度吧；而且加粗也 windows 系统上也是没有效果的，一般都改成了黑体（比如这个模板中成绩页等加粗的地方都是用的黑体），当然你也可以自定义。怎么做，百度吧

9.2.4 公式

所有的符号都要用美元符号包裹 \$，需要用到某一个但是不知道，直接百度，基本上都有

表 9.4 公式

名称	分数	下角标	上角标	矢量	根号	希腊字母	点乘	叉乘	矢量
显示	$\frac{1}{2}$	O_2	a^2	\vec{AB}	$\sqrt[3]{3}$	θ	\cdot	\times	\vec{a}

但是有时候我们只是正文中想用 MoS_2 ，它竟然斜体，不想斜体，我写了个命令，这样用 MoS_2 ，正的吧，常用的命令可以自定义

9.2.5 左边大括号

$$\begin{cases} \vec{e}_1 = \frac{3a}{2}\vec{i} + \frac{\sqrt{3}a}{2}\vec{j} \\ \vec{e}_2 = \frac{3a}{2}\vec{i} - \frac{\sqrt{3}a}{2}\vec{j} \end{cases} \quad (9.1)$$

注意后面有个方程的编号,如果想取消,把上下的两个 *equation* 改成 *equation**

$$\begin{cases} \vec{e}_1 = \frac{3a}{2}\vec{i} + \frac{\sqrt{3}a}{2}\vec{j} \\ \vec{e}_2 = \frac{3a}{2}\vec{i} - \frac{\sqrt{3}a}{2}\vec{j} \end{cases}$$

9.2.6 复杂公式

不会输出的符号,请百度,啥都有

$$\hat{H} = \frac{\epsilon}{2}\hat{\sigma}_z - \frac{\Delta}{2}\hat{\sigma}_x + \sum_k \omega_k \hat{b}_k^\dagger \hat{b}_k + \sum_k \frac{g_k}{2} \hat{\sigma}_z (\hat{b}_k + \hat{b}_k^\dagger) \quad (9.2)$$

9.2.7 等号对齐站

主要是用这个 aligned 放在了方程的环境里,等号前面 & 控制对齐,每一行后面双斜杠换行

$$\begin{aligned} \vec{CH} &= m \cdot \vec{e}_1 + n \cdot \vec{e}_2 \\ &= \frac{3(m+n)a}{2}\vec{i} + \frac{\sqrt{3}(m-n)a}{2}\vec{j} \end{aligned} \quad (9.3)$$

9.2.8 矩阵乘法

其实就是几个 array 组合

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} \quad (9.4)$$

9.2.9 图,并列排

这一句代表这个图片宽度为一行文本宽度的 $\frac{3}{10}$

```
1 width=0.3\textwidth
```



图 9.3 i 兰大易班截图

9.2.10 附页代码

可以在 LZUThesis.clc 里面修改代码格式

java 代码

```
1 System.out.print("i 兰大易班")
2 // 试一下中文注释
```

tex 代码

```
1 width=0.3\textwidth
2 % 注释
```

python 代码

```
1 print("i 兰大易班")
2 # 注释
```

matlab 代码有专门的库，但是没必要高亮太多，而且中文适配有问题，直接按照下面这个就可以

```
1 display("i 兰大易班")
2 % 注释
```

9.2.11 参考文献

这个，百度学术、谷歌学术等网站都可以导出 bibtex 格式的参考文献（知网不行，网上有个人写了个转换器，但是 windows 用不了，就不放了，尽量用谷歌学术把那个文献找出来吧），直接放在 bib/database.bib 文件里、知网需要用其他东西转换，但是我建议用 mendeley 这个软件管理文献，然后可以导出 bibtex 格式的，甚至可以直接复制引用，很方便 [?, ?, ?]。

有些人希望多个参考文献同时引用时用 [1-3] 而不是 [1,2,3]，所以我加了个包 cite。（2020-5-18）

具体怎么用可以百度，我这里告诉你什么可以用，但是具体的，建议百度，更靠谱一些。

有参考文献时，编译要经过 4 步，直接 XeLaTeX → BibTeX → XeLaTeX → XeLaTeX，不然很多问题，我用的 sublime text3，配合插件 LatexTool，直接快快捷键 ctrl - B，就可以自动完成 4 步了，很方便

附录

这里是附录页，附上你的程序或必要的相关知识

编译方式: XeLaTeX → BibTeX → XeLaTeX→XeLaTeX

致 谢

这里是致谢页，你可以在这里致谢你的舍友，老师，朋友，或者我。

论文（设计）成绩表

导师评价你人很好

建议成绩 80

指导教师（签字）



答辩小组意见

优秀

答辩委员会负责人（签字）

成绩 100

学院（盖章）

年 月 日